Project 2

**Brandy Bridges**

April 05, 2010

SSE 658

Dr. Paul E. MacNeil

Contents

[Part 3 Design 4](#_Toc257131770)

[Chapter 11 Architectural Design 4](#_Toc257131771)

[11.1 Architectural Design Decisions 4](#_Toc257131772)

[11.2 System Organization 5](#_Toc257131773)

[11.3 Modular Decomposition Styles 6](#_Toc257131774)

[11.4 Control Styles 7](#_Toc257131775)

[11.5 Reference Architectures 7](#_Toc257131776)

[Chapter 12 Distributed Systems Architectures 8](#_Toc257131777)

[12.1 Multiprocessor Architectures 8](#_Toc257131778)

[12.2 Client-Server Architectures 8](#_Toc257131779)

[12.3 Distributed Object Architectures 9](#_Toc257131780)

[12.4 Inter-Organizational Distributed Computing 9](#_Toc257131781)

[Chapter 13 Application Architectures 9](#_Toc257131782)

[13.1 Data Processing Systems 9](#_Toc257131783)

[13.2 Transaction Processing Systems 9](#_Toc257131784)

[13.3 Event Processing Systems 9](#_Toc257131785)

[13.4 Language Processing Systems 9](#_Toc257131786)

[Chapter 14 Object-Oriented Design 9](#_Toc257131787)

[14.1 Objects and Object Classes 9](#_Toc257131788)

[14.2 An Object-Oriented Design Process 9](#_Toc257131789)

[14.3 Design Evolution 9](#_Toc257131790)

[Chapter 15 Real Time Software Design 10](#_Toc257131791)

[15.1 System Design 10](#_Toc257131792)

[15.2 Real Time Operating Systems 10](#_Toc257131793)

[15.3 Monitoring and Control Systems 10](#_Toc257131794)

[15.4 Data Acquisition Systems 10](#_Toc257131795)

[Chapter 16 User Interface Design 10](#_Toc257131796)

[16.1 Design Issues 10](#_Toc257131797)

[16.2 The UI Design Process 10](#_Toc257131798)

[16.3 User Analysis 10](#_Toc257131799)

[16.4 User Interface Prototyping 10](#_Toc257131800)

[16.5 Interface Evolution 10](#_Toc257131801)

[Part 7 Emerging Technologies 11](#_Toc257131802)

[Chapter 31 Service-Oriented Software Engineering 11](#_Toc257131803)

[31.1 Services as reusable components 11](#_Toc257131804)

[31.2 Service Engineering 11](#_Toc257131805)

[31.3 Software Development with Services 11](#_Toc257131806)

# Part 3 Design

## Chapter 11 Architectural Design

Architectural design is the first stage in the design process. It is the critical link for the design and requirements processes. Architectural design is concerned with building a basic framework that identifies and supports the primary components of a system. Concessions are usually made during the design process to account for system needs affected by certain requirements: performance, security, safety, availability, and maintainability.

### 11.1 Architectural Design Decisions

During the architectural design process, certain decisions must be made about the system being built. The system architects make these decisions based on their experience and knowledge, and they must these critically fundamental questions:

1. Is there a generic application architecture that can act as a template for the system being designed?
2. How will the system be distributed across a number of processors?
3. What architectural style or styles are appropriate for the system?
4. What will be the fundamental approach used to structure the system?
5. How will structural units in the system be decomposed into modules?
6. What strategy will be used to control the operation of the units of the system?
7. How will the architectural design be evaluated?
8. How should the architecture of the system be documented?

Even though every software system is unique, it shares some fundamental domain concepts with other systems that operate in the same application domains. Sommerville uses a product line as an example; he says that a complete product line of software will be built around common core architecture. To me that is evident when we look at a product line we all probably use every day, Microsoft Office. When you use each application in the Microsoft Office product line, they all have the same look and feel and use many of the same functionality to make them easier for those of us who use all the different product.

The ultimate goal of the architectural design process is a design document that will probably include some graphical representations of the design decisions that have been made for the system. These graphical representations are known as architectural models and may include:

1. Static structural models
2. Dynamic process models
3. Interface models
4. Relationship models
5. Distribution models

### 11.2 System Organization

Sommerville states that the organization of a system reflects the basic strategy that is used to structure a system. Sub-system structures directly reflect decisions made about system organization. Sommerville discusses three distinct organizational styles that can be used separately or together:

1. The repository model—this model is best suited for applications that share large amounts of data, and that data is generated by one sub-system but used by other sub-systems. The advantages and disadvantages of the repository model are:
   1. It is an efficient way to share large amounts of data.
   2. Performance may be compromised by the compromise of how the sub-systems agree to share data
   3. Sub-systems that produce data do not “care” how that data is used by other sub-systems.
   4. It may be difficult, or even impossible, for evolution to occur because of the large amounts of information generated by agreed data model.
   5. Backup, security, access control, and recovery are centralized.
   6. Unfortunately, the repository model forces the same policy for the activities in “e” on every sub-system and some sub-systems may have different requirements for those activities.
   7. The repository model makes adding new tools a straight forward process.
   8. It is difficult to distribute the repository over multiple machines.
2. The client-server model – this model has a very simple premise, a client makes a call to a server and waits for a reply. The biggest advantage of the client server model is that it is a distributed architecture and can take advantage of distributed processors. The major components of the client-server model are:
   1. A set of servers that offer services to other sub-systems.
   2. A set of clients that call on services offered by servers.
   3. A network that allows clients to access servers.
3. The layered model – this model organizes a system into layers, each of which provide a set of services. The layered model supports incremental development. The architecture is also easily portable and changeable. A disadvantage of this model is that structuring systems with it can be difficult because certain basic functionality may only be available in the inner layers, and upper layers have to “punch” through to access them. Sommerville suggests that the layered model should incorporate four layers:
   1. Configuration management system layer
   2. Object management system layer
   3. Database system layer
   4. Operating system layer

### 11.3 Modular Decomposition Styles

Modular decomposition styles are concerned with decomposing sub-systems into modules. Sommerville states that there is no clear distinction between sub-systems and modules, but he finds it useful to think of them in the following ways:

1. A sub-system is a system in its own right whose operation does not depend on the services provided by other sub-systems.
2. A module is normally a system component that provides one or more services to other modules and makes use of services available from other modules.

Object-oriented decomposition is a decomposition strategy where you decompose the system into a set of loosely coupled objects that communicate with each other. Function-oriented pipelining is a decomposition strategy where you decompose a system based on functionality where data is input and transformed into output data.

### 11.4 Control Styles

In order for sub-systems to work as a complete system, the sub-systems must be controlled so their services can be delivered to the right place at the right time. Control models are not included in the structural model of a system, instead the architect will organize the sub-systems according to some control model that best supports the structure model that is in use for the system. Sommerville states that there are two generic control styles that are used in software systems:

1. Centralized control – One sub-system has overall responsibility for control and starts and stops the other sub-systems. Centralized control models will fall into one of two categories, the call-return model or the manager model. The call-return model is a top-down model where control starts at the top of the routine hierarchy and, using subroutine calls, passes to lower level of the tree. The strength of the call-return model is it’s rigidity and restricted nature because it is simple to control, however this is also it’s weakness because abnormal processes are very awkward to handle. The manager model is applicable to concurrent systems, where one system acts as the manager of the system manager and controls the stopping and starting of services. The manager model is often used in real-time systems that do not have tight time constraints.
2. Event-based control – Each sub-system can respond to externally generated events that originate from other sub-systems or from the environment of the system. Sommerville discusses two event-driven control models, broadcast models and interrupt-driven models. In broadcast models, an event is broadcast to all sub-systems and any sub-system that is programmed to handle the event will do so. Interrupt-driven models are used exclusively in real-time systems where external interrupts are handled by an interrupt handler, and are then passed to another component for processing.

### 11.5 Reference Architectures

Sommerville defines reference architectures as generic system architecture that is an idealized architecture that includes all the features that systems might incorporate. This is a way of informing designers about the general structure of that class of system. Architectural models that are specific to a particular application domain are called domain-specific architectures. There are two types of domain-specific architectures, generic models and reference models. Generic models are abstractions from a number of real systems that encapsulate the principal characteristics of those systems. Reference models are even more abstract and describe a larger class of systems. Reference architectures are often used as a way to discuss domain-specific architectures and to assess and compare architectural designs.

## Chapter 12 Distributed Systems Architectures

A distributed system is a system where the processing is distributed across several computers instead of being confined to a single machine. Distributed systems can be designed using previously discussed architecture models, but they also have some very specific issues that have to be addressed including resource sharing, concurrency, scalability, and manageability. Distributed systems are usually developed using an object oriented approach, building systems made up of loosely integrated independent parts.

### 12.1 Multiprocessor Architectures

A multiprocessor architecture is the simplest model of a distributed system. Software systems made of multiple processes are not necessarily distributed systems, distribution can be implemented if more than one processor is available, but designers do not always have to consider issues associated with distributed architectures during the design process. Multiprocessor architectures are used primarily when developing large, real-time systems.

### 12.2 Client-Server Architectures

Client-server architectures were previously discussed in chapter 11, but we will go into more detail here. The design of client-server systems should reflect the logical structure of the application. Most times client-server architectures are represented as three separate layers, the presentation layer, the application processing layer, and the data management layer. These layers can overlap in some architecture models, but when designing distributed systems, these layers must be clearly separated because you can then distribute each layer to a different computer. Two-tier client-server architectures for distributed systems normally fall into to two classes:

1. Thin-client – this model has all of the application processing and data management handled on the server and the presentation layer is on the client when the presentation software is run.
2. Fat-client – this model has only the data management portion of the application on the server and all of the application logic and presentation layer on the client machine running the software.

An alternative to the two-tier system is the three-tier system. In the architecture, the presentation, the application processing and the data management are logically separate processes that execute on different processors. The best example of this is the internet. Most internet applications use a client browser to render the presentation layer, then have a web-server that handles the application processing, and finally a database server to handle the data management.

### 12.3 Distributed Object Architectures

Distributed object architectures differ from client-server architectures in the way that services are called. Distributed object architectures remove the distinction between the client and server and the fundamental system components are objects that provide an interface to a set of services provided by those objects. Those services can be called from the client or the server because the objects may be located across multiple computers on a network and will communicate through middleware called a request broker.

Request brokers are required at two levels, at the logical communication level and the component level. Request brokers at the logical communication level is Sommerville’s focus. Sommerville suggests using CORBA standards defined by the Object Management Group (OMG) when building a distributed application. These standards can be downloaded [here](http://www.omg.org/spec/CORBA/3.1/).

The advantages of a distributed object system are:

* Service-providing objects can execute on any node of a network, so decisions about when and how a service is provided can be delayed if needed, and the distinction between fat-client and thin-client models become irrelevant.
* The open architecture allows new resources to be added as needed.
* The system is flexible and scalable.
* The system can be dynamically configured with objects migrating across the network as needed.

An example of a distributed object architecture would be one using web-services. Developers often create applications that make calls to different web-services to access, retrieve, and update data as needed. Those web-services may reside on the same server or different servers. The point is that those web-services are available to the system all the time, regardless of application state.

The biggest disadvantage of distributed object architecture is that it requires a more complex design than the client-server architecture.

### 12.4 Inter-Organizational Distributed Computing

Sommerville discusses two types of inter-organizational distributed computing systems, peer-to-peer architectures and service-oriented systems.

A peer-to-peer system is a distributed system where there is no distinction between clients and servers. Computers in the system can act as both clients and servers. Peer-to-peer applications include file sharing, instant messaging and cooperation support systems. The standards and protocols that enable communications across the network are embedded in the application itself, so every node in the network that takes part in communication must run a copy of the application. Sommerville describes two principal types of network architectures used with peer-to-peer systems:

1. Decentralized architectures – the nodes in the network are not simply functional elements, but are also communication switches that route data and control signals from one node to another.
2. Semi-centralized architectures – the role of the server is to help establish contact between peers in the network or to coordinate the results of the computation.

Service-oriented systems are built using service-oriented architectures and service oriented software engineering methodologies. Both of these are discussed in detail in [Chapter 31.](#_Chapter_31_Service-Oriented)

## Chapter 13 Application Architectures

Generic application architectures are useful when developing systems that have to be customized for different business models. The generic system being developed can then be configured to fit whatever business needs the purchasing customer has. Generic application architectures can be used in numerous ways:

1. As a starting point for the architectural design process
2. As a design checklist
3. As a way of organizing the work of the development team
4. As a means of assessing components for reuse
5. As a vocabulary for talking about types of applications

Sommerville describes four broad types of application architectures, [data-processing applications](#_13.1_Data_Processing), [transaction-processing applications](#_13.2_Transaction_Processing), [event-processing applications](#_13.3_Event_Processing), and [language-processing applications](#_13.4_Language_Processing).

### 13.1 Data Processing Systems

Data processing systems focus on data: storing data, processing data, producing reports and documents based on the data. Data processing systems are batch processing systems where data is input and output in batches from files or databases rather than from a user terminal. Sommerville states that the architecture of batch-processing systems has three major components: an input component, a processing component, and an output component. Data processing systems are function-oriented systems and not object-oriented systems, and as such do not maintain internal state information from one execution to the next. Data-processing systems are fairly simple models, but their data represents the real complexity of the business rules driving the application.

### 13.2 Transaction Processing Systems

A transaction processing system is a system that ensures transactions are processed in such a way so that they do not interfere with each other and so that individual transaction failure does not affect other transactions or the system’s data. From a user perspective, a transaction is any coherent sequence of operations that satisfies a goal. Transaction processing systems are usually interactive systems where users make asynchronous requests for service. Transaction processing systems are usually used for information and resource management systems where the data should be available in real time. With the advent and widespread use of the internet, implementations of information and resource management systems based on internet protocols are the norm, the user interfaces for these systems are built using a web browser. According to Sommerville, e-commerce systems are Internet-based resource management systems that are designed to accept electronic orders for goods or services and then arrange delivery of these goods or services to the customer.

### 13.3 Event Processing Systems

Event processing systems respond to events in the system’s environment or user interface. Sommerville states that the key characteristic of event-processing systems is that the timing of events is unpredictable and the system must be able to cope with these events when they occur. Exception handling and data validation are particularly important in event-processing systems because both can be used to account for the unpredictability of the events of the system.

### 13.4 Language Processing Systems

A language processing system is a system that translates one language to another. Compilers are an example of a language processing system that translates program source code to object code. They accept a natural or artificial language as input, and produce some other representation of that language as an output. Language processing systems are used in situations where the easiest way to solve a problem is to specify that solution as an algorithm or as a description of the system data. Sommerville states that language processing systems have a generic architecture that includes the following components:

1. A lexical analyzer, which takes input language tokens and converts them to an internal form.
2. A symbol table, which holds information about the names of entities used in the text being translated.
3. A syntax analyzer, which checks the syntax of the language being translated. It uses a defined grammar of the language and builds a syntax tree.
4. A semantic analyzer, which uses information from the syntax tree and the symbol table to check the semantic correctness of the input language text.
5. A code generator, which “walks” the syntax tree and generates abstract machine code.

## Chapter 14 Object-Oriented Design

Object oriented design is part of object-oriented development process where an object-oriented strategy is used throughout the development process. Objects created are potentially reusable components because they are independent encapsulations of the state and operations.

### 14.1 Objects and Object Classes

Sommerville defines and object as an entity that has a state and a defined set of operations that operate on that state. Objects are created according to an object class definition. An object class definition is both a type specification and a template for creating objects. Objects talk to one another by requesting services, or calling methods, form other objects and, if necessary, by exchanging the information required for service provision. Classes can be inherited with classes lower down the class hierarchy having the same attributes (properties) and operations (methods) as the parent classes.

Concurrent object implementation can be accomplished in two ways:

1. On the server, where the object is realized as a parallel process with methods corresponding to the defined object operations. Methods start up in response to an external message and may execute in parallel with methods associated with other objects. When they have completed their operation, the object suspends itself and waits for further request for the service.
2. As active objects where the state of the object may be changed by internal operations executing within the object itself. The process representing the object continually executes these operations so never suspends itself.

### 14.2 An Object-Oriented Design Process

Sommerville presents five general stages for the object-oriented design process:

1. Understand and define the context and the modes of the use of the system.
2. Design the system architecture.
3. Identify the principal object in the system.
4. Develop design models.
5. Specify object interfaces

Sommerville wants to make sure we understand that all of the above activities are interlaced, and influential on each other. As such he has a very in depth example of a weather mapping system to illustrate all the different stages.

When taking on a new project, the first thing we have to do is develop an understanding of the relationships between the software that is being designed and its external environment. The system context and the model of the system use represent two complementary models of the relationships between a system and its environment:

* The system context is a static model that describes the other systems in that environment.
* The model of system use is a dynamic model that describes how the system actually interacts with its environment.

The next step we take when developing a system is design the system architecture. In Sommerville’s example, he defines three layers in the software: the interface layer, the data collection layer, and the instruments layer. He also suggests that the best approach to architecture design is to decompose the system so that architectures are as simple as possible.

When designing the architecture is complete, we can move on to identifying the object classes we will need. You can identify necessary object classes in a number of ways:

1. Use grammatical analysis of a natural language description of a system. Objects and attributes are nouns; operations or service are verbs.
2. Use tangible entities (things) in the application domain such as aircraft, roles such as manager, events such as request, interactions such as meetings, locations such as offices, organizational units such as companies, and so on.
3. Use a behavioral approach where the designer first understands the overall behavior of the system; then various behaviors are assigned to different parts of the system and the understanding is derived of who initiates and participates in these behaviors.
4. Use a scenario-based analysis where various scenarios of the system use are identified and analyzed in turn. As each scenario is analyzed, the team responsible for the analysis must identify the required objects, attributes and operations.

Sommerville states that design models show the objects or object classes in a system and, where appropriate, the relationships between these entities. Design models bridge the divide between system requirements and system implementation. There are two types of design models that should be part of the object-oriented design process:

1. Static models – describe the static structure of the system using object classes and their relationships.
2. Dynamic models – describe the dynamic structure of the system and show the interactions between the system objects.

Object interface specification describes the interfaces between the components in the design.

### 14.3 Design Evolution

Changes to any system are inevitable. An important advantage of an object-oriented approach to design is that is simplifies the problem of making changes to the design. The object state does not influence the design, so changing internal details of an object is unlikely to affect any other system objects. Also, because objects are loosely coupled, it is usually straightforward to introduce new objects without significant effects on the rest of the system.

## Chapter 15 Real Time Software Design

Real-time embedded systems are systems that have to respond to and process external events in “real-time”. The correctness of the system does not just depend on what it does, but also on how quickly it does it. Real-time systems are usually organized as a set of cooperating sequential processes. Real-time system architectures are instances of [event-driven architecture.](#_13.3_Event_Processing)

### 15.1 System Design

System design process involves deciding what capabilities are to be implemented in software. Events rather than objects or functions should be central to the real-time software design process. Sommerville describes several interleaved stages in this design process for real-time systems:

1. Identify the stimuli that the system must process and the associated responses.
2. For each stimulus and associated response, identify the timing constraints that apply to both stimulus and response processing.
3. Choose an execution platform for the system: the hardware and the real-time operating system to be used. Factors that influence these choices include the timing constraints on the system, limitations on power available, the experience of the development team and the price target for the delivery system.
4. Aggregate the stimulus and response processing into a number of concurrent processes. A good rule of thumb in real-time system design is to associate a process with each class of stimulus and response.
5. For each stimulus and response, design algorithms to carry out the required computations. Algorithm designs often have to be developed relatively early in the design process to give an indication of the amount of processing required and the time required to complete that processing.
6. Design a scheduling system that will ensure that processes are stated in time to meet their deadlines.

State machine models are a good, language-independent way of representing the design of a real-time system and are an important part of real-time system design methods. A state model of a system assumes that, at any time the system is in one of a number of possible states.

### 15.2 Real-time operating systems

A real-time operating system manages processes and resource allocation in a real-time system. It starts and stops processes so that stimuli can be handled and allocated memory and processor resources. The components for a real-time operating system depend on the size and complexity of the real-time system being developed. Most real-time systems will include the following components:

1. A real-time clock – this provides information to schedule process periodically.
2. An interrupt handler – this manages aperiodic requests for service.
3. A scheduler – this component is responsible for examining the processes that can be executed and choosing one of these for execution.
4. A resource manager—given a process that is scheduled for execution, the resource manager allocated appropriate memory and processor resources.
5. A dispatcher – this component is responsible for starting the execution of a process.

Process management is accomplished using the event-handling processes which must be scheduled for execution in a timely manner to they can detect the event and the processes must be allocated so that appropriate processor resources can be allocated so the processes can meet their deadlines. The process manager has to manage processes with different priorities. The real-time operating system it has to be able to manage at least two priority levels for system processes:

1. Interrupt level – This is the highest priority level. It is allocated to processes that need a very fast response. One of these processes will be the real-time clock process.
2. Clock level – This level of priority is allocated to periodic processes. Periodic processes are processes that must be executed at specified time intervals for data acquisition and actuator control. The process scheduler implements system scheduling policies that determine the order of process execution. The two basic scheduling strategies are:
   * Non pre-emptive scheduling – once a process is scheduled, it will run to completion or until it is blocked for some reason, such as waiting for input.
   * Pre-emptive scheduling – an executing process may be stopped if a higher priority process requires service.

### 15.3 Monitoring and control systems

Monitoring and control systems are important classes of a real-time system. Monitoring and control systems check sensors providing information about the system’s environment and take actions depending on the sensor reading. Monitoring systems take action when some exceptional sensor value is detected; control systems continuously control hardware actuators depending on the value of associated sensors.

Each type of sensor being monitored has its own monitoring process. A monitoring process collects and integrates data before sending it to a control process, which then makes decisions based on the data received and communicates the appropriate control commands to the equipment control processes.

### 15.4 Data acquisition systems

Data acquisitioning systems collect data from sensors for processing and analysis. These systems are used in situations where the sensors are collecting lots of data from the system environment and it isn’t possible or necessary to process the data collected in real-time. These types of systems are commonly used in scientific experiments and process control systems where physical processes like a chemical reaction happen very quickly.

In data acquisitioning systems the sensors may be generating data very quickly and the key problem is to ensure that a sensor reading is collected before the sensor value changes. In real-time systems that involve data acquisition and processing, the execution speeds and periods of acquisition process (the producer) and the processing process (the consumer) may be out of step. When significant processing is required, data acquisition of the system may go faster than the data processing of the system. If only simple computations need to be carried out, the processing may be faster than the data acquisition.

## Chapter 16 User Interface Design

Sommerville defines user interface design as the process of designing the way in which system users access the system functionality and information produced by the is displayed. Our text focuses only on graphical interface design, and does not discuss interfaces that require special displays such as cell phones, DVD players, televisions, and office equipment (like copiers and fax machines). In order for a system to achieve its full potential and be a success, the user interface must be designed to utilize the skills, experience and expectations of the users. Bad user interface design will lead to system users rejecting the system will hinder their efforts rather than enhance them. Sommerville lists several issues to remember when designing user interfaces:

1. People have a limited short-term memory – If you present users with too much information they may not be able take it all in.
2. We all make mistakes – When users make mistakes that trigger warning messages, it creates stress for the user and increases the chances of making operational errors.
3. We have a diverse range of physical capabilities – You should not design for your own capabilities, and assume that users will be able to cope. Some people have better hearing, or vision; some people are color-blind.
4. We have different interaction preferences – Some people like to work with text, others like to work with pictures. The interface should be designed to be as friendly as possible to all potential users.

The preceding issues are the foundation for six user interface design principles:

|  |  |
| --- | --- |
| Principle | Description |
| User familiarity | The interface should use terms and concepts drawn from the experience of the people who will make the most use of the system. |
| Consistency | The interface should be consistent in that, wherever possible, comparable operations should be activated in the same way. |
| Minimal surprise | Users should never be surprised by the behavior of the system. |
| Recoverability | The interface should include mechanisms to allow users to recover from errors. |
| User guidance | The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities. |
| User diversity | The interface should provide appropriate interaction facilities for different types of system users. |

### 16.1 Design Issues

Sommerville says that user interface designers are faced with two essential questions:

1. How should the user interact with the computer system?
2. How should information from the computer system be presented to the user?

User interaction with the system is when commands and data are issued to the system. There are five primary styles of user interaction: direct manipulation, menu selection, form fill-in, command language, and natural language. Direct manipulation usually involves a point device like a mouse that indicates the object to be manipulated and the action. Menu selection is when the user selects a command from a list of possibilities. Form fill-in is when the user fills in fields of a form. Command language is when the user issues a special command and associated parameters to instruct the system what to do. Natural language is when the user issues a command using natural language, but this is usually a front end to a command language interface.

Information presentation refers to how information from the system to the user in a useful way. Sommerville lists five key questions to keep in mind when deciding how to present information:

1. Is the user interested in the precise information or in the relationships between data values?
2. How quickly do the information values change? Should the change in a value be indicated immediately to the user?
3. Must the user take some action in response to a change in information?
4. Does the user need to interact with the displayed information via a direct manipulation interface?
5. Is the information to be displayed textual or numeric? Are relative values of information values important?

You have to be careful not to go too far down one road or another. Presenting information graphically can give the user a nice, easy picture of the information available, but it can be slow and cumbersome on download. Textual representations can be difficult to understand at a glance, but will give the user an abundance of information.

### 16.2 The UI Design Process

The overall UI design process has three core activities: user analysis, system prototyping, and interface evaluation. The user analysis process is where we as developers develop and understanding of the tasks users do, their working environment, the other systems that they use, how they interact with other people in their work, etc. Systems with a diverse group of users have to be designed with this diversity in mind during the UI process. System prototyping can be used to as part of an iterative development process, with users being able to use the system at different points of development and providing feedback. Interface evaluation is where you collect information from users about their actual experiences with the interface. Sommerville focuses on user analysis and interface evaluation, and only a brief discussion of user interface prototyping techniques because he covers system prototyping in depth in a later chapter.

### 16.3 User Analysis

Sommerville says that a challenge for engineers involved in user analysis is to find a way to describe user analyses so that they communicate the essence of the tasks to other designer and to the users themselves. Developers will most likely understand and appreciate the UML designs that result from such analysis, but users will probably find them too technical. The solution is to develop natural language scenarios to describe user activities. User analysis does not generally produce specific user interface requirements, but it helps you understand the needs and concerns of system users and use this in the design process. This understanding means that initial designs are more likely to be accepted by users and get them really involved in the process.

Sommerville lists three basic user analysis techniques: task analysis, interviewing and questionnaires, and ethnography. Task analysis and interviewing focus on the individual and their work. Ethnography takes a broader perspective, and looks at how people interact with others, how they arrange their work environment, and how they come together to find solutions to problems. These techniques are meant to be used together as a complete set of tools to help you in the user analysis portion of the user interface design process, they are complementary approaches that should be used together to help you understand what users do and get insights into what might be an appropriate user interface design.

### 16.4 User Interface Prototyping

Evolutionary or exploratory prototyping with end-user involvement is a practical way to design and develop graphical user interfaces for software systems. User involvement in the design process is essential to the success of the system. Users, like the rest of us, have a hard time mentally visualizing a potential interface and critiquing it. It is much easier to see some examples, and then give feedback on likes and dislikes of the examples. Sommerville gives us three approaches to use for user interface prototyping:

1. Script-driven approach – In this approach, you create screens with visual elements and allow the user to interact with them.
2. Visual programming languages – With this approach, you use a visual programming language to build a prototype using the reusable components available in the language itself.
3. Internet-based prototyping – This approach makes use of web browsers to offer a ready-made user interface. It is the fast way to develop user interface prototypes.

### 16.5 Interface Evaluation

Sommerville defines interface evaluation as the process of assessing the usability of an interface and checking that it meets user requirements. The nature of interface evaluation means that it should be part of the normal verification and validation process for software systems. Metrics can be developed for interface evaluation, but since the usability of the interface can be subjective you usually have to use your own judgment and experience in the process. Usability attributes include:

1. Learnability – How long does it take a new user to become productive with the system?
2. Speed of operation – How well does the system response match the user’s work practice?
3. Robustness – How tolerant is the system of user error?
4. Recoverability – How good is the system at recovering from user errors?
5. Adaptability – How closely is the system tied to a single model of work?

# Part 7 Emerging Technologies

## Chapter 31 Service-Oriented Software Engineering

Sommerville states that service-oriented architectures are a way of developing distributed systems where the components of these systems are stand-alone services. Because the services can be distributed anywhere in the world, standard protocols are used in the communications and information exchanges. Web services are the dominant service-oriented architecture in use today. The key standards for web service-oriented architectures are:

1. SOAP – The message interchange standard that supports the communication between services.
2. WSDL – The Web Service Definition Language standard that defines the way service providers should define the interface to these services.
3. UDDI – The Universal Description, Discovery, and Integration standard that defines the components of a service specification that may be used to discover the existence of a service.
4. WS-BPEL – A standard for a workflow language that is used to define process programs involving several different services.

### 31.1 Services as reusable components

Services are not exactly the same as the components when considering reusability because they do not the “requires” interface that defines what other components must be present, service “requires” interfaces define what functionality is available from the service. Services do not make method calls to other services the way that traditional system components make method calls to other software components.

Service descriptions are expressed in WSDL with a specification that defines “what” the service does, “how” it communicates, and “where” to find it.

When using Microsoft Visual Studio 2008, we access web services by adding a web reference to our projects. The integrated development environment (IDE) makes accessing and communicating with web services easy during the development process. The IDE automatically generates all the XML documents required to support the communications and exchanges of information.

### 31.2 Service Engineering

Sommerville defines service engineering as the process of developing services for reuse in service-oriented applications and states that is has many commonalities with component engineering. The engineers have to make sure that the service is a reusable abstraction that will be useful to different systems, they must develop generally useful functionality and they have to document the service so that potential users can both discover and understand the service. There are three logical stages in the service engineering process:

1. Service candidate identification – the process where you identify possible services that should be built and define those service requirements. This requires a comprehensive analysis and understanding of the organizations business processes. The goal of service candidate identification is to identify services that are logically coherent, independent and reusable.
2. Service design – the process where you design the logical and WSDL service interfaces. A major point to remember during the design process is that services are stateless and will have no impact on the management of the state of applications that subscribe to the service. Managing the state of those applications is the application responsibility. There are three stages of interface design, logical interface design, message design and WSDL development. Logical interface design begins with requirements and defines the operation names and parameters associated with the service. Exceptions and how the exceptions are handled should also be fined at the logical interface design stage.
3. Service implementation and deployment – the process where you actually implement the service, test it, and make it available for use. The implementation will most likely involve programming using a standard programming language like Java or C#, both of which include extensive libraries with support for service development. Services may also be implemented using legacy systems (to be discussed in the next paragraph). Once the implementation is complete, the service must be tested before it is put into a production environment. Testing the service requires examining and partitioning the inputs, creating input messages that reflect input combinations and then validating that the outputs are as expected. Deployment is the last stage of the process of service development. Most software installed on servers today makes the deployment process very easy; you just have to install the file that contains an executable service in the appropriate directory.

Sommerville suggests that one of the most important uses of services is to build “wrappers” that make functionality from legacy systems available via service oriented architectures. Imagine you are the federal government and have been using the same legacy system to track the money used to repair items on contract since the 1970’s. All of a sudden, every one wants to be able to access the data from via an internet browser. Obviously, the legacy system does not have this option. A solution is to wrap the legacy functionality in component, and expose it via a web service. Now, different “web” systems can subscribe to the web service and access the legacy data that was once unavailable in those environments.

### 31.3 Software Development with Services

Sommerville says that the development of software using services is based around the idea that you compose and configure services to create new, composite services. These may be integrated with a web user interface to create a web application or may be used as components in some other service composition. Service composition is used to integrate separate business processes to provide an integrated process offering extensive functionality. Sommerville provides us with six key stages in the process of service construction by composition:

1. Formulate outline workflow – In this stage you use the requirements for the composite service as a basis for creating an “ideal” service design.
2. Discover services – In this stage you search service registries to discover what services exist, who provides them, and the details of the service provision.
3. Select possible services – In this stage you select the possible services that can implement workflow activities from the set of service candidates.
4. Refine workflow – In this stage you refine the workflow based on the information about the services you selected.
5. Create workflow program – In this stage the abstract workflow design is transformed into an executable program.
6. Test completed service or application – In this stage, the testing of the completed, composite service is done.

Sommerville states that workflow design is a design process that involves analyzing existing or planned business processes to understand the different stages of these processes then representing the process being design in a workflow design notation. Workflows represent business process models and are usually represented graphically.

Sommerville states that testing is important in all system development processes to demonstrate that a system meets its functional and non-functional requirements and to detect defects that have been introduced during the development process. There are however differences between typical software testing and service testing, one of which is that since services are offered by external providers, source code of the implemented service is not available to testers. Listed below are further difficulties that testers may face when testing services:

1. External services are under the control of the service provider rather that the user of the service and the service provider may withdraw these services at any time, or make changes to them that can affect previous testing experiences.
2. The long-term vision of service-oriented architectures is for services to be bound dynamically to service-oriented applications.
3. As, in most cases, a service is available to different customers, the nonfunctional behavior of that service is not simply dependent on how it is used by the application that is being tested.
4. The payment model for services could make service testing very expensive.
5. The notion of compensation actions that are invoked where some exception occurs and previous commitments that have been made have to be revoked. Ensuring that these services actually fail during the testing process can be very difficult.

These problems are very severe when external services are used, but the risk is mitigated when the service is used within the same company or if the corporations trust the services offered by their partners. In those cases the source code may be available to guide the testing process.